# Técnicas para Aumento de Performance de Núcleos de Computação Intensiva encontrados em Simulações de Escoamentos Multifásicos

**Virginia S. Costa**
Faculdade de Engenharia Mecânica - UERJ
Rio de Janeiro, RJ
E-mail: virscosta@gmail.com

**Luiz Mariano Carvalho**
Departamento de Matemática Aplicada, IME - UERJ
Rio de Janeiro, RJ
E-mail: luizmc@ime.uerj.br

**Norberto Mangiavacchi**
Faculdade de Engenharia Mecânica - UERJ
Rio de Janeiro, RJ
E-mail: norberto.mangiavacchi@gmail.com

**Resumo:** *Nos últimos anos, o surgimento de novas tecnologias na área da computação tem possibilitado que diversos ramos da ciência desenvolvam técnicas de simulação de fenômenos físicos. Um desses ramos, em especial, é a dinâmica dos fluidos, que estuda a física de processos que envolvem escoamentos dos fluidos. Assim, a partir de aproximações das equações que modelam estes fenômenos, obtém-se sistemas de equações cujas dimensões e cujo padrão de esparsidade variam de acordo com as técnicas de discretização utilizadas e de acordo com o domínio contínuo representado. Assim, tendo como objeto de estudo estes grandes sistemas lineares, o presente trabalho se propõe a analisar técnicas computacionais de solução e aproximação, dedicando especial atenção aos núcleos de computação intensiva como, por exemplo, multiplicações matriz-matriz.*

## 1 Motivação

Esta seção é baseada em [15], e tem como objetivo servir de motivação para os estudos conduzidos neste trabalho, dentro do contexto da dinâmica de fluidos. Será utilizada uma abordagem Lagrangeana-Euleriana Arbitrária (ALE, ou seja, *Arbitrary Lagrangian-Eulerian*), onde as equações de *Navier-Stokes* são desenvolvidas em um referencial se movendo no espaço com velocidade arbitrária. Quando este referencial se move com a mesma velocidade do fluido, obtém-se a descrição Lagrangeana, e quando o referencial está parado, obtém-se a descrição Euleriana. Assim, considerando-se os domínios material ($S_{\boldsymbol{\xi}}$), referencial ou computacional ($S_{\boldsymbol{\eta}}$) e espacial ($S_{\mathbf{x}}$), mostrados na Figura 1, na descrição Lagrangeana-Euleriana arbitrária, os domínios $S_{\boldsymbol{\xi}}$ e $S_{\boldsymbol{\eta}}$ estão se movendo enquanto $S_{\mathbf{x}}$ está parado. Considerando-se que uma partícula $\boldsymbol{\xi}$ de fluido que ocupa o domínio $S_{\boldsymbol{\xi}}$ se move

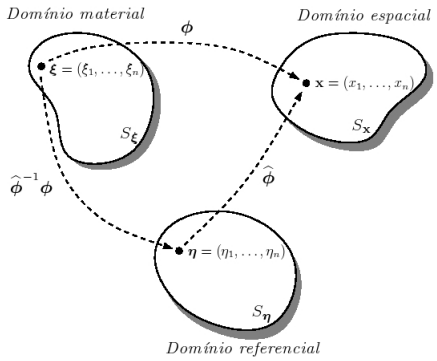


Figura 1: Domínio utilizados na formulação ALE.

para um ponto $\mathbf{x}$ em um dado instante $t$ no domínio espacial, o movimento de um fluido pode ser descrito por uma transformação $\boldsymbol{\phi}$ tal que $\mathbf{x} = \boldsymbol{\phi}(\boldsymbol{\xi}, t)$. Assim, define-se *velocidade* da partícula $\boldsymbol{\xi}$ no domínio espacial como

$$\mathbf{u} = \left.\frac{\partial \mathbf{x}}{\partial t}\right|_{\boldsymbol{\xi}} = \frac{\partial}{\partial t}\,\boldsymbol{\phi}(\boldsymbol{\xi}, t).$$

Da mesma forma, seja $\widehat{\boldsymbol{\phi}}$ a transformação que descreve o movimento de um ponto $\boldsymbol{\eta}$ no domínio $S_{\boldsymbol{\eta}}$ para um ponto $\mathbf{x}$ no domínio $S_{\mathbf{x}}$, então define-se velocidade do ponto $\boldsymbol{\eta}$ no domínio espacial como

$$\hat{\mathbf{u}} = \left.\frac{\partial \mathbf{x}}{\partial t}\right|_{\boldsymbol{\eta}} = \frac{\partial}{\partial t}\,\widehat{\boldsymbol{\phi}}(\boldsymbol{\eta}, t).$$

Seja $f$ uma propriedade expressa como função de $(\boldsymbol{\xi}, t)$ ou $(\mathbf{x}, t)$. Considerando-se as seguintes derivadas temporais

$$\left.\frac{\partial}{\partial t} f(\mathbf{x}, t)\right|_{\boldsymbol{\xi}} = \left.\frac{\partial f}{\partial t}\right|_{\mathbf{x}} + (\mathbf{u} \cdot \nabla) f, \qquad (1)$$

$$\left.\frac{\partial}{\partial t} f(\mathbf{x}, t)\right|_{\boldsymbol{\eta}} = \left.\frac{\partial f}{\partial t}\right|_{\mathbf{x}} + (\hat{\mathbf{u}} \cdot \nabla) f, \qquad (2)$$

subtraindo-se 2 de 1, tem-se

$$\frac{Df}{Dt} = \left.\frac{\partial f}{\partial t}\right|_{\boldsymbol{\xi}} = \left.\frac{\partial f}{\partial t}\right|_{\boldsymbol{\eta}} + ((\mathbf{u} - \hat{\mathbf{u}}) \cdot \nabla)\, f$$

que é a *derivada material* ou *substantiva* da propriedade $f$ em um referencial $\boldsymbol{\eta}$ que se move com velocidade $\hat{\mathbf{u}}$.

A partir da descrição Lagrangeana-Euleriana arbitrária, com o auxílio dos Teoremas de Leibnitz e de Gauss, é possível descrever as equações de *conservação de massa* e *conservação de quantidade de movimento linear*, também chamadas de *Equações de Navier-Stokes*, da seguinte forma (adimensional),

$$\frac{\partial(\rho \mathbf{u})}{\partial t} + ((\mathbf{u} - \hat{\mathbf{u}}) \cdot \nabla)\rho\mathbf{u} = -\nabla p + \frac{1}{N^{\frac{1}{2}}} \nabla \cdot \left[\mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T)\right] + \rho \mathbf{g} + \frac{1}{E_0}\mathbf{f}$$
$$\nabla \cdot \mathbf{u} = 0,$$

onde $N = \frac{\rho_0^2 D^3 g}{\mu_0^2}$ e $E_0 = \frac{\rho_0 g D^2}{\sigma_0}$, são o *Número de Galileu* e o *Número de Eötvös*, respectivamente, e onde $\rho_0$, $\mu_0$ e $\sigma_0$ são, respectivamente, valores de referência para massa específica ($\rho$), viscosidade ($\mu$) e coeficiente de tensão superficial ($\sigma$), $D$ é o diâmetro (aparece em escoamento com bolhas) e $g$ é a força gravitacional. A força $\mathbf{f}$ definida por $\mathbf{f} = -\sigma\kappa\nabla H$, onde $\sigma = \sigma(\mathbf{x}, t) \in \mathbb{R}$ é o coeficiente de tensão interfacial entre os fluidos, $\kappa = \kappa(\mathbf{x}, t) \in \mathbb{R}$ denota a curvatura local da superfície que define a interface e $H : \mathbb{R}^m \longrightarrow \{0, 1\}$ é uma função de *Heaviside* que pode ser definida como 1 onde existe um determinado fluido ou 0 caso contrário, representa, no caso de escoamentos multifásicos, uma força campo que está ligada à tensão interfacial, e portanto, deve agir diretamente na interface entre dois fluidos.

Discretizando-se as equações de *Navier-Stokes*, em sua formulação variacional, no espaço (através do método de Taylor-Galerkin) e em relação ao tempo por um sistema semi-implícito (veja [15] para maiores detalhes), obtem-se o seguinte sistema de equações

$$\begin{bmatrix} \mathbf{B} & -\Delta t\mathbf{G} \\ \mathbf{D} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}^{n+1} \\ \boldsymbol{p}^{n+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}^n \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \boldsymbol{bc}_1 \\ \boldsymbol{bc}_2 \end{bmatrix},$$

onde agora o sistema é escrito apenas para as incógnitas do problema, ou seja, $\boldsymbol{u}^{n+1} = [u_1^{n+1}, \ldots, u_{Nu}^{n+1}, v_1^{n+1}, \ldots, v_{Nv}^{n+1}]^T$, $p^{n+1} = [p_1^{n+1}, \ldots, p_{Np}^{n+1}]^T$, sendo $Nu$, $Nv$ e $Np$ o número de incógnitas (nós livres) para velocidade na direção $x$, velocidade na direção $y$ e pressão respectivamente. A notação para as matrizes e vetores foi mantida a mesma por simplicidade. A matriz $\mathbf{B}$ é dada por $\mathbf{B} = \mathbf{M}_\rho + \frac{\Delta t}{Re}\mathbf{K}$ que é simétrica e positiva-definida, e o lado direito representa as grandezas conhecidas no tempo $n$, $\boldsymbol{r}^n = -\Delta t\left(\mathbf{A}\boldsymbol{u}^n - \frac{1}{Fr^2}\mathbf{M}_\rho \boldsymbol{g} - \frac{1}{We}\mathbf{M}\boldsymbol{f}\right) + \mathbf{M}_\rho \boldsymbol{u}^n$ , mais as condições de contorno que nada mais são do que as contribuições dos valores conhecidos de velocidade e pressão no lado direito do sistema. Utilizando-se a *decomposição LU* ([11], [12] e [13]) para a separação da velocidade e pressão no sistema acima, tem-se

$$\begin{bmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{D} & \Delta t\mathbf{D}\mathbf{B}^{-1}\mathbf{G} \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\Delta t\mathbf{B}^{-1}\mathbf{G} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}^{n+1} \\ \boldsymbol{p}^{n+1} \end{bmatrix} =$$

$$= \begin{bmatrix} \boldsymbol{r}^n \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \boldsymbol{bc}_1 \\ \boldsymbol{bc}_2 \end{bmatrix}$$

Uma aproximação comum para a inversa $\mathbf{B}^{-1}$ pode ser considerar $\mathbf{B}^{-1} = \mathbf{M}_\rho^{-1}$ [2], e portanto,

o sistema ainda pode ser escrito como

$$\mathbf{B}\tilde{\boldsymbol{u}} = \boldsymbol{r}^n + \boldsymbol{b}\boldsymbol{c}_1 \quad (3)$$

$$\Delta t \mathbf{D}\mathbf{M}_\rho^{-1}\mathbf{G}\boldsymbol{p}^{n+1} = -\mathbf{D}\tilde{\boldsymbol{u}} + \boldsymbol{b}\boldsymbol{c}_2 \quad (4)$$

$$\boldsymbol{u}^{n+1} = \tilde{\boldsymbol{u}} + \Delta t \mathbf{M}_\rho^{-1}\mathbf{G}\boldsymbol{p}^{n+1} \quad (5)$$

onde procedimento para a solução das equações é dado na seguinte ordem:

1. Resolve-se $\tilde{\boldsymbol{u}}$ de (3);
2. Resolve-se $\boldsymbol{p}^{n+1}$ de (4);
3. Encontra-se a velocidade final $\boldsymbol{u}^{n+1}$ usando (5);

Como se pode notar, nos passos acima relacionados, encontram-se sistemas lineares que podem ser simétricos e positivos-definidos e sistemas que podem ser não simétricos. Para a resolução destes sistemas é aconselhável o uso de métodos iterativos e pré-condicionadores. Os métodos iterativos devido às grandes dimensões das matrizes envolvidas - o que torna inviável a solução por métodos diretos - e os pré-condicionadores para melhorar o condicionamento destas matrizes. Para o sistema simétrico e positivo-definido, por padrão, se usa o método Gradiente Conjugado Pré-condicionado, ou PCG (*Preconditioned Conjugate Gradient*[5]), expresso no Algoritmo 1.

**Algoritmo 1** *Gradiente Conjugado Pré-condicionado*

1. Entrar com $x^{(0)}$
2. Calcular $r^{(0)} = b - \mathbf{Ax^{(0)}}$
3. Enquanto critério de convergência não for atingido faça
4. resolva $\mathbf{Mz^{(i-1)}} = r^{(i-1)}$
5. $\rho_{i-1} = {r^{(i-1)}}^T z^{(i-1)}$
6. Se $i = 1$
7. $p^1 = z^{(0)}$
8. senão
9. $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
10. $p^{(i)} = z^{(i-1)} + \beta_{(i-1)}p^{(i-1)}$
11. fim-se
12. $q^{(i)} = \mathbf{Ap^{(i)}}$
13. $\alpha_i = \rho_{i-1}/{p^{(i)}}^T q^{(i)}$
14. $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
15. $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
16. Atualizar critério de convergência
17. fim-enquanto

Já para os demais sistemas, pode ser usado o Método do Resíduo Mínimo Generalizado, ou GMRES (*Generalized Minimum Residual Method* [14]), ou o Método do Gradiente Biconjugado Estabilizado, ou Bi-CGSTAB (*Biconjugate Gradient Stabilized* [16]), também pré-condicionados, como mostram os Algoritmos 2 e 3. Note que, nos três algoritmos, existem alguns elementos em negrito, a fim de se enfatizar a presença de sistemas lineares triangulares e das *multiplicações matriz-vetor*, que neste trabalho, juntamente com as *multiplicações matriz-matriz*, são chamadas de NCI, ou seja, *núcleos de computação intensiva*.

**Algoritmo 2** *Algoritmo do GMRES pré-condicionado pela esquerda*

1. Calcule $r_0 = \mathbf{M^{-1}(b - Ax_0)}$, $\beta = ||r_0||_2$, e $v_1 = r_0/\beta$
2. Para $j = 1, \cdots, m$, Faça
3. Calcule $w = \mathbf{M^{-1}Av_j}$
4. Para $i = 1, \cdots, j$, Faça
5. $h_{i,j} = (w, v_i)$
6. $w = w - h_{i,j}v_i$
7. fim-para
8. Calcule $h_{j+1,j} = ||w||_2$ e $v_{j+i} = w/h_{j+i,j}$
9. fim-para
10. Defina $V_m = [v_1, \cdots, v_m]$,
11. Defina $\hat{H}_m = h_{i,j\,1\leqslant i\leqslant j+1, 1\leqslant j\leqslant m}$
12. Calcule $y_m = argmin_y ||\beta_{e_1} - \mathbf{\hat{H}_m y}||_2$
13. Calcule $x_m = x_0 + \mathbf{V_m y_m}$
14. Se o critério de convergência for satisfeito Pare
15. Senão calcule $x_0 = x_m$ e volte para a linha 1.

**Algoritmo 3** *Algoritmo BI-CGSTAB pré-condicionado*

1. Calcule $r_0 = b - \mathbf{Ax^{(0)}}$ a partir de um $x^{(0)}$ escolhido
2. Escolha $\tilde{r}$ ($\tilde{r} = r^{(0)}$ por exemplo)
3. Para $i = 1, \cdots$ Faça
4. $\rho_{i-1} = \tilde{r}^T r^{(i-1)}$
5. Se $\rho_{i-1} = 0$ o método falha
6. Se $i = 1$
7. $p^{(i)} = r^{(i-1)}$
8. Senão
9. $\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\omega_{i-1})$
10. $p^{(i)} = r^{(i-1)} + \beta_{i-1}(p^{(i-1)} - \omega_{i-1}v^{(i-1)})$
11. fim-se
12. Resolva $\mathbf{M\hat{p}} = p^{(i)}$
13. $v^{(i)} = \mathbf{A\hat{p}}$
14. $\alpha_i = \rho_{i-1}/\tilde{r}^T v^{(i)}$
15. $s = r^{(i-1)} - \alpha_i v^{(i)}$
16. Verifique a norma de $s$
17. Se $||s||$ for pequena o suficiente
18. $x^{(i)} = x^{(i-1)} + \alpha_i \hat{p}$ e Pare
21. Resolva $\mathbf{M\hat{s}} = s$
22. $t = \mathbf{A\hat{s}}$
23. $\omega_i = t^T s/t^T t$
24. $x^{(i)} = x^{(i-1)} + \alpha_i \hat{p} + \omega_i \hat{s}$
25. $r^{(i)} = s - \omega_i t$
26. Verifique a convergência; continue se necessário
27. Para continuar é necessário que $\omega_i \neq 0$
28. fim-para

O objetivo deste texto é dar um tratamento mais adequado aos NCI e, para isto, testou-se as operações matriz-vetor (por aparecer constantemente em algoritmos como os relacionados acima) e matriz-matriz (onde

se gasta muito tempo e recurso de hardware). Assim, seja $A$ um matriz densa de números randômicos, de ordem $n$, e $x$ um vetor de $n$ elementos randômicos, para se testar a multiplicação matriz-vetor, repetiu-se a operação $Ax$ por 1000 (mil) vezes e, para se testar a multiplicação matriz-matriz, repetiu-se a operação $A^2$ por 10 (dez) vezes, utilizando-se, respectivamente, as subrotinas DGEMV (D=double, GE=general, MV=matrix-vector) e DGEMM (D=double, GE=general, MM=matrix-matrix), oriundas de um conjunto de subrotinas básicas de álgebra linear (BLAS, *Basic Linear Algebra* [1]) disponíveis em diferentes abordagens. Algumas dessas abordagens podem ser vistas na seção a seguir.

# 2 BLAS

As subrotinas BLAS (Basic Linear Algebra Subprograms) são rotinas originalmente escritas em Fortran 77, mas que hoje estão disponíveis nas principais linguagens e adaptadas às principais arquiteturas e compiladores existentes, e que oferecem meios para que operações básicas com matrizes e vetores sejam executadas. Este conjunto de subrotinas está dividido em:

- Nível 1 - executa operações escalar-vetor e vetor-vetor [10];
- Nível 2 - executa operações matriz-vetor [4];
- Nível 3 - executa operação matriz-matriz [3].

Apesar de os sistemas vistos na seção 1 serem esparsos, levando-se em consideração que uma matriz esparsa pode ser divida em blocos que, dependendo do tamanho, podem ter a maior parte de seus elementos não-nulos e que existem pacotes de Álgebra Linear orientados a objeto que, mesmo resolvendo problemas esparsos, utilizam as subrotinas densas do BLAS (como o Trilinos [9], por exemplo), esta seção se dedica a mostrar teoricamente quatro distribuições de BLAS voltadas ao tratamento de problemas densos[1].

[1] A MKL também é voltada para problemas esparsos. No entanto, considera-se aqui somente as multiplicações matriz-vetor e matriz-matriz densas

## 2.1 ACML

A ACML (AMD Core Math Library) é uma biblioteca desenvolvida pela AMD® que consiste dos seguintes itens:

- A plena implementação do Nível 1, 2 e 3 Basic Linear Algebra Subroutines (BLAS), com as principais rotinas otimizadas para alta performance em processadores AMD Opteron [2].
- Um pacote completo de Álgebra Linear (LAPACK) rotinas.
- Transformadas Rápidas de Fourier (FFTs), precisão simples, dupla e dados complexos.
- Geradores de números aleatórios em precisão simples e dupla.
- Suporte a OpenMP.

## 2.2 MKL

A MKL (Math Kernel Library) é a biblioteca desenvolvida pela Intel® que apresenta os seguintes recursos:

- BLAS, LAPACK, ScaLAPACK e rotinas auxiliares.
- Solvers Iterativos e Diretos.
- Suporte a Multi-Thread e outras.

## 2.3 GotoBLAS

Nas últimas décadas, alguns projetos de grupos de pesquisa no mundo todo têm perseguido alta performance no que diz respeito a operações básicas de Álgebra Linear. Tipicamente, estes projetos vêm organizando a computação em torno de um "núcleo interno", $C = A^T B + C$ (considerado aqui como NCI), que guarda um dos operadores na cache L1, enquanto mantêm parte dos outros operadores fora desta cache. Algumas variações incluem abordagens que utilizam este princípio para vários níveis de cache ou que aplicam o mesmo princípio à cache L2, enquanto ignoram a L1. O objetivo principal destas abordagens é amenizar o custo de

[2] Neste trabalho, as distribuições de BLAS não serão testadas em processadores AMD Opteron, por não haver disponibilidade destes no laboratório de teste.

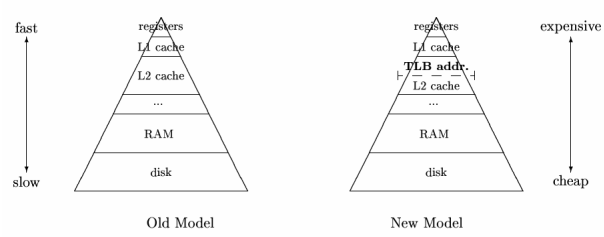


Figura 2: Considerações básicas sobre arquitetura.

se mover dados através dos diversos níveis de memória.

O GotoBlas [6] é uma biblioteca desenvolvida pelo Centro de Computação Avançada do Texas, Universidade de Austin que apresenta uma abordagem diferente, observando que, para a geração atual de arquiteturas de computadores, parte dos custos com a computação deste núcleo interno está associada às perdas na TLB (*Translation Look-aside Buffer Table*). Embora seja levada em consideração a importância das caches, o fato de maior relevância nesta abordagem é a minimização dessas perdas na TLB ([6], [7] e [8]).

### 2.4 ATLAS

O ATLAS, isto é, Automatically Tuned Linear Algebra Software [17], é um projeto de pesquisa cujo objetivo principal é aplicar técnicas empíricas para garantir portabilidade às Rotinas de Álgebra Linear em termos de performance.

Há projetos (e o ATLAS pode ser citado como exemplo), que propõem novos paradigmas para a produção de rotinas de alta performance. Este paradigma é conhecido como *AEOS* [17], isto é, *Automated Empirical Optimization of Software*.

Em pacotes baseados na AEOS são oferecidos vários métodos de se fazer as operações requisitadas e, a partir destas requisições, é feita empiracamente uma análise de tempo para se decidir qual dos métodos oferecidos é o melhor para determinado tipo de arquitetura.

Neste contexto, o ATLAS usa geradores de código (i.e., programas que escrevem programas) para oferecer diferentes métodos de se resolver uma dada operação e, além disto, apresenta sofisticados scripts de busca e mecanismos robustos de medição de tempo para se encontrar o melhor método para uma determinada arquitetura.

## 3 Testes e Resultados

Nesta seção, serão feitos testes de comparação de tempo (em segundos) gasto para se executar as subrotinas DGEMV e DGEMM, conforme descrito no último parágrafo da seção 1, utilizando-se as distribuições vistas na seção 2. Nestes testes, foram utilizados computadores com as seguintes característica:

**Computador 1:** Processador: Intel Core 2 Duo, modelo E6600, 2.4 GHz, 4 MB de L2; Memória RAM: 2 GB, DDR-2/533 MHz; Sistema Operacional: Linux (Ubuntu 8.04). Os resultados para o computador 1 estão expressos nas Tabelas 1 e 2.

| Dimensão ($\times 10^3$) | ACML | ATLAS | BLAS | Goto | MKL |
|---|---|---|---|---|---|
| 1,0 | 3,00 | **1,37** | 4,76 | 3,08 | 1,61 |
| 2,0 | 13,52 | **6,98** | 19,03 | 14,41 | 7,26 |
| 3,0 | 31,92 | **15,79** | 42,79 | 32,66 | 16,19 |
| 4,0 | 58,03 | **28,04** | 76,04 | 57,69 | 28,76 |
| 6,0 | 131,01 | **63,67** | 170,89 | 129,87 | 64,68 |
| 7,0 | 175,69 | **87,16** | 232,67 | 177,66 | 88,73 |

Tabela 1: Tempo (em segundos) de Execução da rotina DGEMV.

| Dimensão ($\times 10^3$) | ACML | ATLAS | BLAS | Goto | MKL |
|---|---|---|---|---|---|
| 1,0 | 4,64 | 2,70 | 47,52 | 2,65 | **2,50** |
| 2,0 | 37,40 | 21,54 | 381,50 | **18,72** | 18,87 |
| 3,0 | 125,05 | 72,39 | 1282,50 | 67,38 | **62,68** |
| 4,0 | 299,80 | 171,06 | 3045,30 | 151,84 | **145,53** |
| 6,0 | 1008,20 | 580,09 | 10337,00 | 501,49 | **488,83** |
| 7,0 | 1593,50 | 915,88 | 16439,00 | 842,59 | **777,50** |

Tabela 2: Tempo (em segundos) de Execução da rotina DGEMM.

As Tabelas 1 e 2 (assim como, as tabelas a seguir), têm como objetivo comparar o BLAS padrão (Netlib), amplamente utilizado em resolução de sistemas lineares, com as demais distribuições de BLAS. Neste contexto, a biblioteca MKL foi a que apresentou os melhores tempos para a multiplicação matriz-matriz (DGEMM - Tabela 2). Já na multiplicação matriz-vetor (DGEMV - Tabela 1), o ATLAS apresentou resultados melhores, porém bem próximos aos da MKL. Agora, será feito um teste com um processador AMD.

**Computador 2:** Processador: AMD Athlon X2, modelo 4.200+, 2.2 GHz, 1 MB de L2; Memória RAM: 4 GB, DDR/400 MHz; Sistema Operacional: Linux (Debian Etch 4.0).

| Dimensão ($\times 10^3$) | ACML | ATLAS | BLAS | Goto | MKL |
|---|---|---|---|---|---|
| 1,0 | 5,39 | **2,77** | 3,88 | 5,29 | 3,00 |
| 2,0 | 21,33 | **10,99** | 15,48 | 20,80 | 12,26 |
| 3,0 | 47,42 | **25,88** | 34,74 | 46,35 | 27,71 |
| 4,0 | 83,67 | **45,47** | 61,83 | 81,89 | 48,92 |
| 6,0 | 187,63 | **104,51** | 142,69 | 182,84 | 111,71 |
| 7,0 | 254,59 | **139,61** | 195,20 | 248,43 | 151,05 |

Tabela 3: Tempo (em segundos) de Execução da rotina DGEMV.

| Dimensão ($\times 10^3$) | ACML | ATLAS | BLAS | Goto | MKL |
|---|---|---|---|---|---|
| 1,0 | **5,40** | 5,42 | 39,27 | 9,37 | 6,65 |
| 2,0 | **41,18** | 42,24 | 312,67 | 65,07 | 48,07 |
| 3,0 | **137,42** | 141,33 | 1052,45 | 218,50 | 155,38 |
| 4,0 | **323,28** | 334,89 | 2485,30 | 508,73 | 375,61 |
| 6,0 | **1080,87** | 1553,05 | 8638,07 | 1728,61 | 1232,99 |
| 7,0 | **1712,03** | 2360,11 | 13880,99 | 2759,26 | 1918,63 |

Tabela 4: Tempo (em segundos) de Execução da rotina DGEMM.

Na Tabela 3, observa-se que a biblioteca ATLAS apresentou o melhor desempenho na multiplicação matriz-vetor. É importante notar, ainda na Tabela 3, os tempos obtidos pelas bibliotecas ACML e GOTO em relação ao BLAS padrão (Netlib). Percebe-se que, nesta configuração (computador 2), essas bibliotecas levaram mais tempo para executar a operação $Ax$. Já na multiplicação matriz-matriz, Tabela 4, a biblioteca ACML apresentou os melhores tempos.

É importante lembrar que as bibliotecas aqui utilizadas foram compiladas com as orientações básicas dos desenvolvedores, exceto a biblioteca BLAS padrão que foi obtida a partir dos repositórios do Ubuntu e do Debian. É importante ressaltar também que os programas que utilizaram a ACML e a ATLAS foram compilados com a opção `- fopenmp`, o que utilizou a biblioteca GotoBLAS foi "linkado" com a biblioteca `-lpthread` e o que utilizou a MKL utilizou as bibliotecas `-lmkl_em64t`, `-lguide` e `-lpthread`, conforme as orientações dos respectivos desenvolvedores.

# 4 Conclusão

Neste trabalho, a partir de sistemas lineares oriundos de simulações de escoamentos multifásicos, observou-se que operações de multiplicação matriz-vetor estão presentes com freqüência em algoritmos de resolução bastante utilizados, consumindo uma quantidade de tempo significativa ao serem executadas. Este fato leva a um trabalho mais direcionado a estas operações que, juntamente com as operações de multiplicação matriz-matriz, foram chamadas de NCI (*núcleos de computação intensiva*). Levando-se em consideração que existem muitos softwares - em C/C++ e outras linguagens - que há tempos utilizam um conjunto de surotinas básicas de Álgebra Linear (BLAS) padrão para o tratamento dos NCI e que existem outras distribuições de BLAS disponíveis, houve a necessidade de testá-las a fim de se saber qual seria a melhor opção. Assim, foram feitos testes (seção 3) para se comparar a diferença de tempo de execução de duas subrotinas do BLAS: a DGEMV (multiplicação matriz-vetor de precisão dupla) e a DGEMM (multiplicação matriz-matriz de precisão dupla). A subrotina DGEMV foi testada por aparecer com freqüência nos algoritmos de resolução e a subrotina DGEMM por consumir muitos recursos de hardware. A partir desses testes, foi possível perceber que o desempenho de determinadas distribuições varia dependendo do tipo de máquina, podendo ser muito bom em uma arquitetura e muito ruim em outra. Percebeu-se também que o desempenho pode varia de acordo com a subrotina executada, isto é, uma distribuição pode apresentar um bom desempenho na subrotina DGEMM e um desempenho ruim na DGEMV. Isto significa que, ao se escolher uma distribuição de BLAS, deve-se considerar a arquitetura/microarquitetura do computador e também qual operação será executada para este computador.

# Referências


[1] BLAST Forum. *Basic linear algebra subprograms techical (BLAST) forum*. University of Tennessee, August 2001.

[2] W. Chang, F. Giraldo, and B. Perot. Analysis of an exact fractional step

method. *Journal of Computational Physics*, 180:183–199, 2002.

[3] J. Dongarra, J. DuCroz, I. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft.*, 16:1–17, 1990.

[4] J. J. Dongarra, J. Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–17, 1988.

[5] G. H. Golub and C. F. Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)*. The Johns Hopkins University Press, October 1996.

[6] K. Goto and R. Geijn. On reducing TLB misses in matrix multiplication. Technical Report TR-2002-55, The University of Texas at Austin, Department of Computer Sciences, 2002. FLAME Working Note #9., 2002. (ftp://ftp.cs.utexas.edu/pub/techreports/tr02-55.ps.gz).

[7] K. Goto and R. van de Geijn. High performance implementation of the level-3 BLAS. *ACM Transactions on Mathematical Software*, 35(1). Also available in www.cs.utexas.edu/users/flame/pubs/TOMS_GOTO2.pdf.

[8] K. Goto and R. A. van de Geijn. Anatomy of a high-performance matrix multiplication. *ACM Transactions on Mathematical Software*, 34(3), 2008. Also available in www.cs.utexas.edu/users/flame/pubs/GOTO_TOMS_FINAL.pdf.

[9] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.

[10] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *Application for Computing Machinery Transactions on Mathematical Software*, 5:308–323, 1979.

[11] M. J. Lee, B. D. Oh, and Y. B. Kim. Canonical fractional-step methods and consistent boundary conditions for the incompressible navier-stokes equations. *Journal of Computational Physics*, 168:73–100, 2001.

[12] M.-J. Ni, S. Komori, and N. Morley. Projection methods for the calculation of incompressible unsteady flows. *Numerical Heat Transfer*, 44:553–551, 2003.

[13] J. B. Perot. An analysis of the fractional step method. *Journal of Computational Physics*, 108:51–58, 1993.

[14] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[15] F. Simeoni. *Simulação de Escoamentos Multifásicos em Malhas não Estruturadas*. PhD thesis, Universidade de São Paulo, São Carlos,, Agosto 2005.

[16] H. A. van der Vorst. BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992.

[17] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (www.netlib.org/lapack/lawns/lawn147.ps).